TREBALL DE FI DE CARRERA

TÍTOL: Obtenció de codis correctors d'errors mitjançant algorismes evolutius

TITULACIÓ: Enginyeria Tècnica de Telecomunicació, especialitat Sistemes de Telecomunicació

AUTOR: Laura Sala Dot

DIRECTORS: Francesc Comellas Padró i Marta Serra Llanas

DATA: Juny 2005

Títol: Obtenció de codis correctors d'errors mitjançant algorismes evolutius

Autor: Laura Sala Dot

Directors: Francesc Comellas Padró i Marta Serra Llanas

Data: Juny 2005

Resum

La generació de codis correctors d'errors constitueix un aspecte bàsic de les telecomunicacions. L'objectiu principal d'aquest TFC és l'aplicació d'un algorisme evolutiu anomenat "formigues" per a l'obtenció de codis correctors d'error binaris de pes constant. En aquesta memòria, en primer lloc es fa una introducció breu als codis correctors d'errors i als principals algorismes d'optimització combinatòria. A continuació, es presenta una adaptació de l'algorisme "formigues" per a aconseguir codis binaris de longitud n, distància de Hamming d, i pes constant w i amb un nombre de paraules, A(n,d,w), el més gran possible. S'expliquen els detalls de funcionament del mètode i finalment es comparen els resultats obtinguts mitjançant l'algorisme amb d'altres aconseguits per altres algorismes com la recuita simulada (simulated annealing) i els algorismes genètics.

Title: Design of binary error correcting codes using evolutionary algorithms

Author: Laura Sala Dot

Directors: Francesc Comellas Padró, Marta Serra Llanas

Date: June 2005

Overview

The design of error correcting codes constitutes an important part of telecommunications theory. The main aim of this TFC is the use of an evolutionary algorithm (ants) to obtain constant weight binary error correcting codes. After a short introduction to error correcting codes and to some combinatorial optimization algorithms we present an adaptation of the "ants" multi-agent algorithm to obtain binary codes of length n, Hamming distance d, and constant weight w with a total of words, A(n,d,w), as large as possible. After providing the details of the method we compare the results obtained with this algorithm with those obtained with simulated annealing and genetic algorithms.

AGRAÏMENTS

Vull dedicar aquest projecte integrament als meus pares i a la meva parella que m'han ajudat en tot moment, no tant sols en donar-me suport per tirar endavant amb aquest TFC sinó per comprendre'm i ajudar-me quan m'ha fet falta al llarg dels meus estudis. Sempre han estat al meu costat i sempre han confiat en mi i en les ganes que tenia de fer realitat aquest somni: gràcies família.

També volia donar les gràcies al meu director Francesc Comellas per ajudar-me quan ho he necessitat i per estar sempre disponible. També donar les gràcies a la meva coordinadora per l'empenta que m'ha donat per tirar endavant.

Finalment dir que aquests tres anys han estat molt amens gràcies a la companyia de gent com la Núria, la Maribel, l'Eva Aranda, l'Eva González i altres amics que he anat coneixent i apreciant. Gràcies a tots de tot cor.

ÍNDEX

INTRODUCCIÓ

Una bona part de les Telecomunicacions actuals es dedica a l'estudi i al desenvolupament de la transmissió de dades. La informació que volem transmetre es pot expressar en un conjunt de dades que enviem per un canal que sovint canvia segons les nostres necessitats. El que esperem quan enviem un missatge és que la persona que el rep el pugui reproduir com si fos l'original. Però normalment això no és així. Els canals de comunicacions porten intrínsec un soroll que malmet les comunicacions. O bé la informació original es distorsiona, és a dir, el missatge que rep no és el mateix que l'original, o bé el missatge s'acaba perdent. Degut a tots aquets factors explicats anteriorment és fa imprescindible el desenvolupament de noves tècniques que millorin les condicions actuals.

Existeixen molts tipus de codis i no tots són vàlids per aquesta fi. Els codis correctors d'errors són una solució bastant òptima pel problema que se'ns presenta. Aquests codis es conceben per a la detecció i/o correcció d'errors. Encara que desitjaríem eradicar totalment els errors en les comunicacions, aquest és un fet gairebé impossible, encara que és important continuar amb la recerca de nous codis per intentar disminuir, si és possible, els errors que es produeixen. Per tal de trobar aquests codis hem realitzat un algorisme informàtic basat en el model teòric que permet trobar el que busquem en un temps limitat.

Hem cregut convenient fer una introducció als codis així com explicar l'algorisme formigues per després poder introduir-nos dins el problema d'obtenció de codis amb les bases teòriques establertes. Un cop tenim aquests coneixements calia explicar com apliquem nosaltres aquest algorisme sobre codis correctors d'errors per acabar obtenint codis de pes constant amb N paraules on N ha de ser el més gran possible. Finalment els resultats i les conclusions són fonamentals per veure si aquest algorisme és prou bo.

Després de veure'n els resultats doncs, la conclusió principal és que es comporta d'una manera similar a d'altres però se n'ha d'acabar de comprovar el bon funcionament amb proves més complexes.

2. INTRODUCCIÓ ALS CODIS

2.1. Coneixements previs

Abans d'aprofundir en el tema de generació de codis binaris de pes constant creiem convenient fer una petita introducció als coneixements previs teòrics necessaris per tal de poder tractar correctament el problema.

2.1.1. Codis correctors d'errors

El concepte de codi no és pas nou ja que un codi ens permet transmetre informació amb fiabilitat i/o seguretat i l'intercanvi d'informació és bàsic en les relacions humanes. Fa molt de temps que existeix, per exemple, el conegut codi Morse o més recentment ha aparegut el codi ASCII, entre molts d'altres.

El codi Morse va sorgir a mans de Samuel Morse cap l'any 1832 quan va començar la construcció del telègraf que anys després el portaria a guanyar més d'un premi. Morse creia que els pobles tenien una necessitat de comunicació i per aquest motiu va impulsar el seu invent fins que va aconseguir que es produís amb èxit la primera transmissió de dades. Posteriorment es va procedir a cobrir el territori americà amb línies telegràfiques. El codi ASCII té un origen més recent però uns objectius semblants al Morse. ASCII (*A*merican *S*tandard *C*ode for *I*nformation *I*nterchange) és un codi de caràcters basat en l'alfabet llatí. Va ser creat l'any 1963 per ASA (comitè americà d'estàndards) com una evolució dels conjunts de codis utilitzats fins llavors en telegrafia. Actualment el codi Morse ha quedat gairebé obsolet mentre que el codi ASCII està en ple rendiment. Els sistemes informàtics utilitzen aquest codi per poder representar textos i per al control de dispositius que utilitzen textos.

La transferència de dades a través d'un canal sovint comporta la pèrdua parcial d'aquestes dades. Els canals de transmissió estan caracteritzats per un soroll implícit que comporta aquesta pèrdua. Tenint el problema localitzat, ara el que es preua és un disseny acurat d'un sistema que permeti la recuperació de la informació original per part del receptor.

Un codi corrector d'errors ideal és aquell que permet que la informació que transmet un emissor arribi correctament al receptor. De totes maneres és gairebé impossible obtenir un codi ideal ja que per a què es donés el cas hauríem de transmetre molta redundància, tanta o més de la informació que tenim. Com que aquesta solució no es viable (gastaríem molta energia, temps i recursos) els codis que utilitzem són el resultat de la investigació en aquest camp i són capaços de corregir la major part d'errors que es produeixen en la transmissió de dades i el senyal original pot reproduir-se correctament en el receptor.

El senyal original és codificat amb un codi corrector d'errors en la part emissora, mentre que la part receptora quan rep el senyal distorsionat pel canal és capaç de detectar els errors i, en alguns casos, d'eliminar-los gràcies a aquesta codificació.

Els codis més utilitzats són els codis de Hamming, els codis de Trellis i els codis convolucionals. Tots ells proporcionen una fiabilitat semblant.

2.1.2. Codis binaris correctors d'errors. Definicions.

2.1.2.1. Codis binaris de pes constant i longitud n

Un codi,C, està format per *N* paraules diferents *p*. Cada paraula conté una combinació de *n* bits {0,1}. Es defineix el pes d'una paraula com el número de {1} que conté. Per tant, si parlem de pes constant, vol dir que cadascuna de les paraules que formen el codi té un pes determinat i igual a les altres paraules.

$$C = \{ p_1, p_2, ..., p_n \}$$

Exemple:

Paraula 1.- 10111000011100100001
Paraula 2.- 01100100010000011111
Paraula N.- 00100110100101101010

Aquest codi conté *N paraules* amb un pes constant de *W=9* i una longitud de *n=20* bits.

2.1.2.2. Distancia de Hamming

La distancia de Hamming del codi és una altra característica a tenir en compte. Es defineix entre dues paraules qualssevol del codi com el número de bits que difereixen entre elles. En l'exemple anterior la distancia de Hamming entre la paraula 1 i 2 és de DH=12.

En el disseny de codis binaris correctors d'errors però, el que ens importa és la distancia de Hamming mínima que es defineix com a:

$$dh \min = \min DH(p1,p2) \text{ on } \begin{cases} p1 \neq p2 \\ p1, p2 \in C \end{cases} \quad \textbf{(2.1)}$$

El concepte de la distancia de Hamming mínima és rellevant perquè ens indica el número d'errors que es poden detectar(e_d) i/o corregir(e_c).

$$dh\min \geq e_d + 1$$
$$dh\min \geq 2e_c + 1 \qquad \textbf{(2.2)}$$

Seguint l'exemple anterior i havent exposat tots els coneixements previs, podem expressar les característiques d'un codi determinat com:

A *(n,d,w) = N* paraules. Per exemple podem tenir un codi de A (9,4,4) = 5 paraules. Això vol dir que aquest codi està compost per 5 paraules i que cada paraula conté 9 bits. El pes de la paraula (i per tant del codi) és de 4 i la distancia de Hamming mínima és també de 4.

2.2. Generació de codis binaris de pes constant

La generació de codis es pot fer de diverses maneres. Com ja hem explicat, en aquest cas, les paraules d'un codi qualsevol A (n,d,w) no tenen relació directe.

A l'hora de generar les paraules del codi s'han de tenir en compte certes restriccions que imposa el codi *A (n,d,w)*. De fet, es tracta de generar un codi d'*N* paraules que cadascuna tingui *n* bits binaris i que totes elles compleixin un pes constant *w.* Per tant, una paraula d'*n* bits ha de contenir *w* uns que estan distribuïts dins de la paraula de forma aleatòria i sense seguir cap estereotip.

Taula 2.3. Exemple d'un codi que conté paraules correctes i paraules incorrectes

Codi *A (10,5,4)*	
Paraules correctes	Paraules incorrectes
0010100101	0100010010
1100000011	1011101000
0101010010	0200100101

Cal destacar que la distància de Hamming no intervé a l'hora de generar les paraules del codi.

2.2.1. Associació d'un codi a un graf.

L'algorisme de les formigues és un sistema potent per trobar solucions a diferents problemes no trivials com per exemple l'assignació de freqüències en xarxes de Telecomunicacions i la coloració de grafs.

Tal com ja hem explicat, necessita una estructura predefinida i estàtica per poder actuar que acostuma a ser un graf. Aquest ha estat el primer problema a resoldre, trobar un graf estàtic a partir del codi aleatori generat. Es requereix d'un estudi per determinar de quina manera es pot realitzar aquesta associació i quines restriccions i limitacions se'ns presenten.

El més lògic semblava associar cada node d'un graf amb una paraula del codi però els codis que volem buscar no contenen totes les paraules possibles sinó només un subconjunt escollit de forma aleatòria. Ens trobem davant un problema.

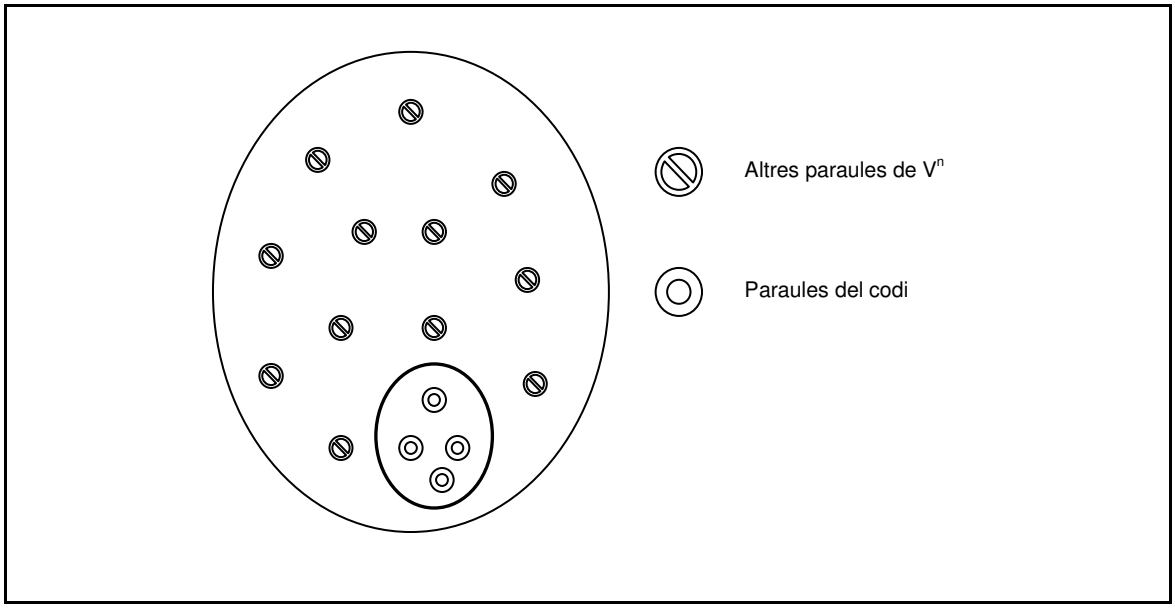

Fig.2.4 Espai V^n que conté totes les paraules d'un codi

Per solucionar el problema la mesura adaptada ha estat associar una paraula del codi a un node sense tenir en compte que hi haurà moltes paraules que es quedaran sense representació nodal.

El problema de com associar un codi a un graf està parcialment resolt. De totes maneres, un graf conté arestes i són aquestes les que hem d'incorporar per tal que les paraules del codi estiguin connectades entre elles. Sense arestes, les formigues són incapaces de desplaçar-se. Per tant, aquesta part és vital en el disseny del graf. De fet, diferents topologies poden donar resultats diferents i fiabilitats diferents. Cal doncs, provar quina topologia és més adequada per a què les formigues puguin actuar més eficaçment.

2.2.1.1. Topologia Aleatòria

La topologia aleatòria és aquella que no es regeix per cap mètode ni per cap teoria a l'hora de connectar dos nodes. Les variables d'aquesta topologia són el número d'arestes per cada node i els nodes que formen l'aresta. Amb aquestes variables és gairebé impossible reproduir dos grafs iguals (com més nodes hi hagi més impossible es fa).

De totes maneres, a l'hora de plantejar la implementació d'aquesta topologia sorgeixen problemes teòrics que cal solucionar.

El fet de generar aleatòriament les arestes pot fer que un node o conjunt de nodes quedin disconnexs o bé formin un subconjunt que farien que acabéssim tenint dos grafs enlloc d'un. Aquest sistema no és viable perquè les formigues serien incapaces de passar d'un graf a l'altre per la falta d'una aresta que els connecti.

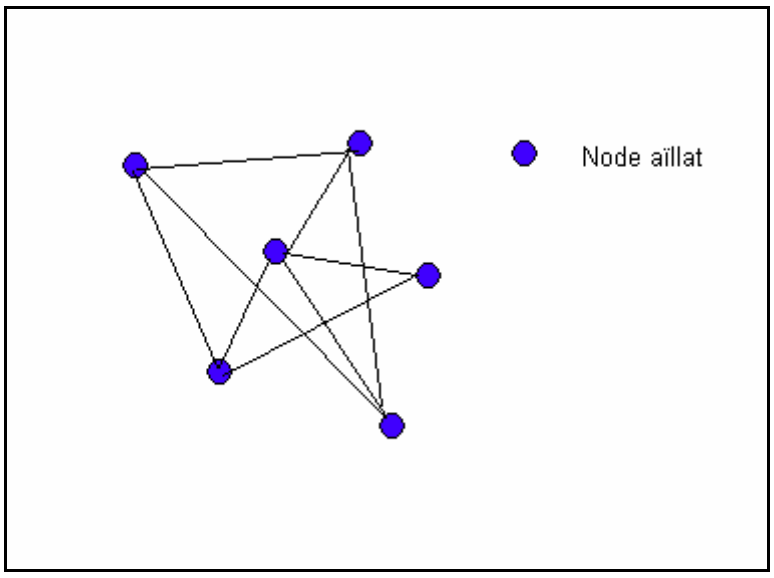

Fig.2.5 Node disconnex al graf principal degut a l'elecció aleatòria de les arestes

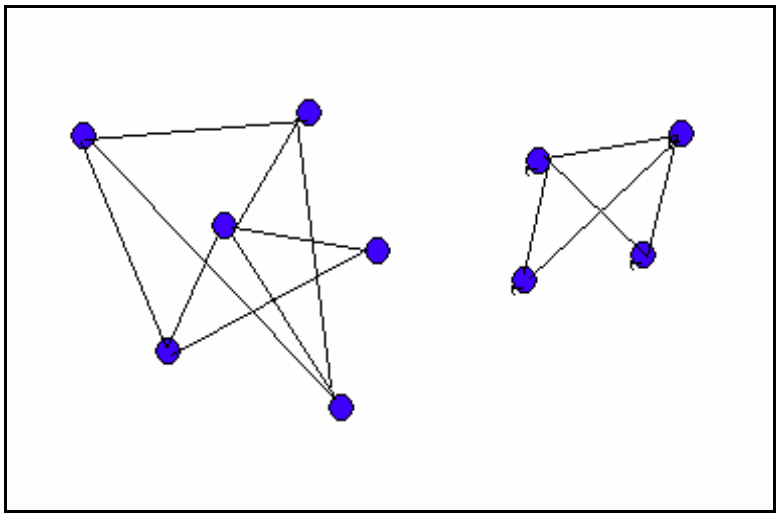

Fig.2.6 Grafs disconnexs per falta d'una aresta aleatòria que els connecti

Un altre factor que cal tenir en compte és que una aresta és bidireccional. És a dir, que està definida un cop i entre dos nodes i no es pot repetir. A l'hora de generar les arestes cap tenir-ho en compte perquè sinó podríem tenir dos arestes per comunicar els mateixos nodes i això faria treballar amb més dificultat i inútilment a les formigues.

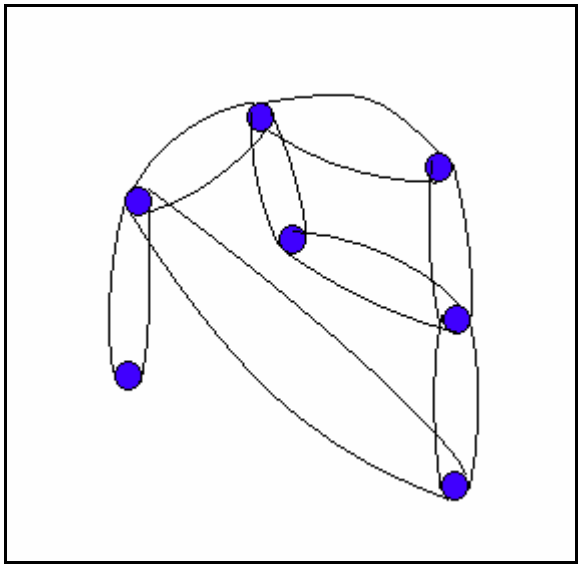

Fig.2.7 Graf amb arestes bidireccionals

Taula 2.8. Exemple d'un codi que genera un graf amb una topologia aleatòria.

	A (6,2,3) = 7
1	001101
2	001010
3	100101
4	110010
5	010101
6	111000
7	000111

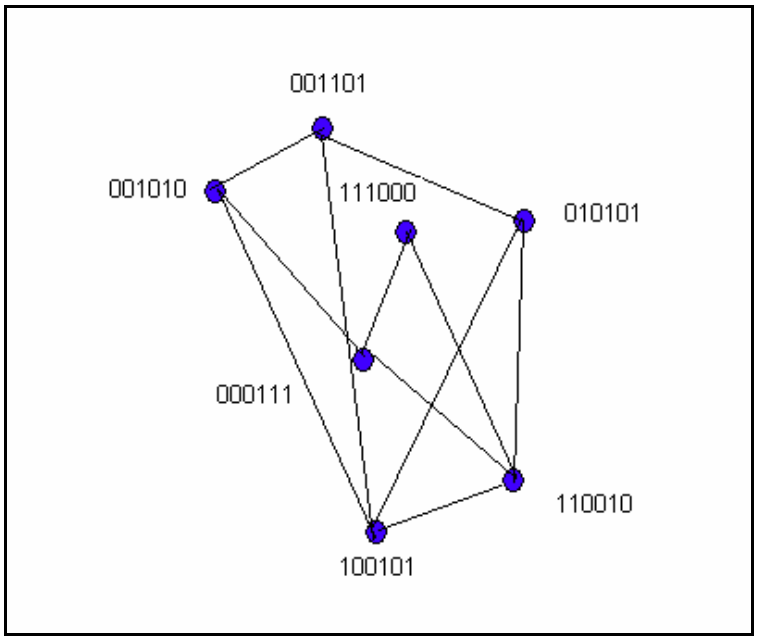

Fig.2.9 Graf generat aleatòriament a partir del codi de la Taula 2.8

Les paraules que formen aquest codi que serveix d'exemple compleixen la distancia de Hamming mínima que en aquest cas és dos. Per tant l'exemple seria d'un codi on ja hi ha actuat les formigues i han trobat una solució final satisfactòria: totes les paraules compleixen la DH mínima. L'exemple es podria haver donat sense que es complís aquesta condició.

2.2.1.2. Topologia 1/3

La topologia 1/3 és una topologia circular i la seva nomenclatura (1/3) ens indica quins nodes de posició relativa estan connectats amb un node qualsevol x. És a dir, si agafem de referència un node x, l'u ens indica que x està connectat amb el node següent al seu (sempre en sentit de les agulles del rellotge) i amb el node de tres posicions més enllà. Cada node per tant, té quatre arestes.

Aquesta topologia sí que es regeix doncs per una forma concreta i donat un determinat número de nodes sempre serà la mateixa. És fàcil doncs, reproduir-la. A més a més no hi ha cap restricció a l'hora de crear-la ja que és impossible que el graf associat tingui cap subconjunt o node disconnex.

Seguint amb l'exemple 2.8, l'associació d'un cert codi a un graf utilitzant aquesta topologia per establir les connexions quedaria representada de la manera següent:

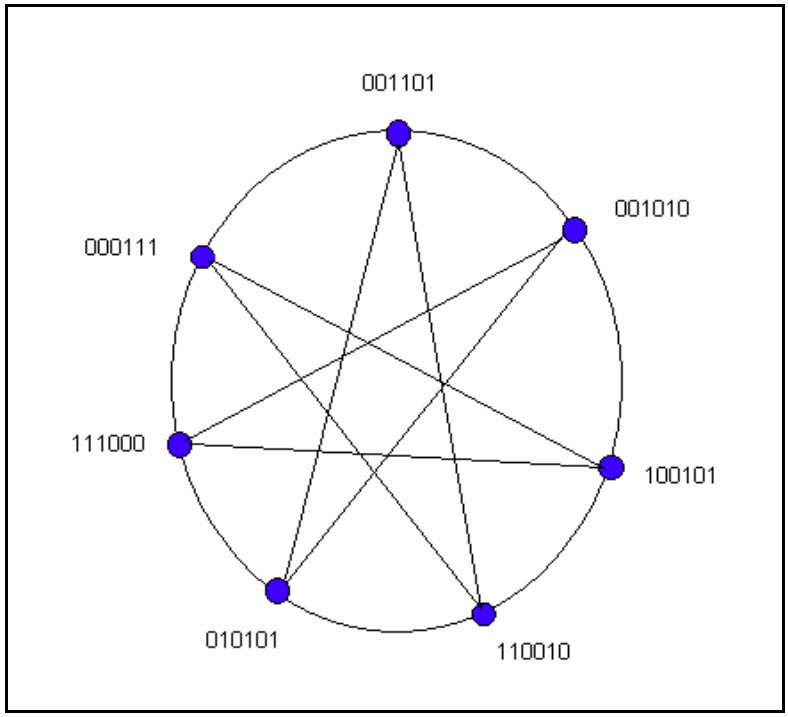

Fig.2.10 Graf generat a partir de la topologia 1/3 segons el codi de la Taula 2.8

3 . ALGORISMES EVOLUTIUS PER A GENERAR CODIS CORRECTORS D'ERRORS

3.1. Algorismes evolutius

Existeix una classificació per determinar quan un problema té solució i quin és el cost associat a aquest. Una de les classes més rellevants d'aquesta classificació és la dels problemes coneguts com NP-Complets. Són els més difícils de solucionar i per aquest motiu calen mètodes de resolució aproximats. Com que la dimensió del problema és massa gran, els algorismes exhaustius són inviables ja que escrutar totes les possibilitats comportaria un cost desorbitat. Per altra banda s'ha pogut comprovar que per aquest tipus de problemes, els algorismes més eficaços són els inspirats en la natura, com per exemple els algorismes genètics, simulated annealing i actualment, també s'està experimentant amb el mètode formigues.

El que s'intenta fer, doncs, alhora de solucionar un problema d'aquestes característiques és, primer de tot, intentar solucionar-lo directament i de forma lògica. Un cop vist que el problema és complex i té un cost temporal elevat cal aplicar altres mètodes que intentin aproximar-se a la realitat per tal que el cost sigui menor encara que perdem exactitud en el resultat. Quan apliquem aquest cas podem utilitzar altres mètodes com per exemple els heurístics o els metaheurístics.

Els algorismes heurístics són aquells que basant-se en la realitat o sentit comú ens poden aportar una solució raonable al problema encara que aquesta, no sigui ni de bon tros la desitjada. Però a canvi, obtenim una solució de manera força ràpida i una solució més independent de la qualitat de les dades originals.

Com a mètodes heurístics podem trobar la cerca aleatòria, la partició, la construcció, la cerca local, els mètodes aproximats, entre d'altres.

La cerca aleatòria, com explica el seu nom, consisteix en explorar aleatòriament l'espai d'estats i retornar la solució que s'ha trobat que compleix un cost mínim. Aquest mètode no assegura la qualitat de la solució.

La partició consisteix en separar el problema en parts més petites per tal que siguin més fàcilment resolubles i arribar així a una solució global degut a la unió de cada una de les subsolucions. Tot i així, no és fàcil fraccionar el problema i tampoc ho és poder-ne unir les subsolucions.

La construcció es basa en un conjunt de normes que permeten anar construint la solució final pas a pas. Trobar aquestes normes no sempre és fàcil sobretot si tenim en compte que el problema es pot estancar degut, precisament, a alguna d'aquestes regles.

La cerca local consisteix en explorar un espai de solucions limitat i intentant trobar la que garanteixi menor cost de les escrutades. El problema és de fonaments: si la cerca és local no es pot garantir que la millor solució es trobi a l'espai explorat.

Els mètodes aproximats costen en modificar lleugerament els models matemàtics extrapolats dels problemes en qüestió i modificar-los per a què siguin més fàcils. Potser en aquest cas els models deixen de complir certes restriccions. El que s'espera d'aquest mètode és una solució que sigui satisfactòria i que a més a més hagi complert les restriccions originals.

Els mètodes hetaheurístics intenten no estancar-se en trobar una solució tal i com els pot passar als heurístics. Per tal de complir aquesta perspectiva, aquests mètodes adjunten una sèrie de variables que permeten una major flexibilitat de l'algorisme.

En els algorismes metaheurístics s'inclouen els algorismes genètics, simulated annealing (recuita simulada), tabu search (cerca tabu), algorismes multiagent, entre altres.

3.1.1. Algorismes Genètics

El primer en tractar aquest tipus d'algorismes va ser John Holland l'any 1950. A partir de llavors va començar a desenvolupar les seves idees i a formar-se per tal d'expandir els seus coneixements. Va ser cap l'any 1960 quan dins el grup *Logic of Computers* de la *Universitat de Michigan* en *Ann Arbor*,on hi donava classes, la seva recerca va començar a donar fruits. A partir de llavors i juntament amb altres científics del seu entorn i alumnes de la mateixa Universitat van començar a treballar en aquest camp de manera més activa. Més endavant, cap 1975, David Goldberg va conèixer a Holland i va passar a ser el seu pupil. Com que Goldberg treballava d'enginyer Industrial va intentar aplicar els A.G. en aquest camp i en va aconseguir un cert èxit ja que va ser capaç de programar un A.G. amb un ordenador.

Els A.G. són mètodes sistemàtics per a la resolució de problemes de cerca i optimització que apliquen els mètodes de l'evolució biològica com la selecció basada en la població, la mutació i la reproducció sexual. Intenten trobar $(x_1,...x_n)$ tals que $F(x_1,...x_n)$ sigui màxim. Després de parametritzar el problema en una sèrie de variables $(x_1,...,x_n)$ aquestes es codifiquen en un cromosoma. Tots els operadors utilitzats en un A.G. s'aplicaran sobre les poblacions de cromosomes o també de manera individual.

Un A.G. és independent del problema per la qual cosa el converteix amb un mètode robust i alhora dèbil perquè no està especialitzat en cap.

Les solucions codificades en un cromosoma competeixen per veure quina d'elles és millor solució (que no vol dir que sigui la millor de les possibles). A més a més, només aquells que resolguin millor el problema sobreviuran i

podran donar el seu material genètic a les generacions posteriors, tal i com passa amb l'evolució de les especies animals. La diversitat genètica s'introdueix amb les mutacions i les reproduccions sexuals.

Una breu descripció dels passos que segueix aquest algorisme pot ser la següent:

1. Generar a l'atzar una població inicial amb M individus
2. Repetir durant G generacions:
   2.1. Avaluar els individus
   2.2. Guardar la millor solució
   2.3. Per tots els individus:
      2.3.1. Fer un sorteig entre la població anterior per triar 2 pares. Els individus amb millor cost han de tenir una probabilitat més alta de ser triats.
      2.3.2. Seleccionar a l'atzar un punt de creuament
      2.3.3. Creuar els dos pares per obtenir el nou individu
      2.3.4. Si random()<probabilitat de mutació, mutar aquest individu

Per tant, un algorisme genètic consisteix en trobar de quins paràmetres depèn el problema, codificar-los en un cromosoma i aplicar els mètodes de l'evolució: selecció i reproducció sexual amb intercanvi d'informació i alteracions que generen diversitat.

3.1.2. Simulated annealing

Simulated annealing és una tècnica que s'utilitza per intentar trobar una solució als problemes d'optimització intentant variacions a l'atzar de la solució actual. Es va proposar l'any 1953.

Les variacions no sempre són bones. Una pitjor variació s'accepta com a nova solució amb una probabilitat p_b, que disminueix a mesura que segueix l'execució. Com més petit és l'índex de disminució, més probable és que l'algoritme trobi una solució òptima o propera.

L'algorisme es pot resumir de la següent manera:

1. Generar aleatòriament una solució inicial. Fixar la temperatura inicial, $T_k=T_o$.

2. Repetir N_k vegades:
   2.1. Modificar lleugerament (de forma aleatòria) la solució i calcular la seva funció de cost.
      2.1.1. Si és millor, acceptar-la com a nova solució.
      2.1.2. Si és pitjor, acceptar-la solament si $e^{\frac{\Delta f}{T_k}} < rand$
      on Δf és la diferència de cost entre la millor solució i la que s'està revisant i Tk és la temperatura actual.

3. Disminuir T_k i repetir 2 fins que $T_k<T_{min}$

Simulated annealing s'ha utilitzat en varis problemes com per exemple en sistemes termodinàmics per tal de convergir la solució final i en problemes combinatoris. On ha estat més eficient ha estat en el disseny de circuits.

3.1.3. Algorismes multiagent

Els més coneguts són els algorismes A.C.O. (Ant Colony Optimization) que són models inspirats en el comportament de colònies de formigues reals. Fóren proposats inicialment per *Colorni*. Més endavant *Maniezzo* va desenvolupar el que coneixem com a algorisme de les formigues basat en ferormones i *Dorigo* i els seus col·laboradors l'han aplicat al problema del viatjant i d'altres problemes combinatoris obtenint bons resultats. La idea principal d'aquest algorisme iteratiu és la cerca en paral·lel que els agents (formigues) duen a terme simulant fins i tot el rastre de ferormones que deixen per indicar cap on són les bones solucions. La característica principal es que les formigues es mouen en l'espai d'estats de totes les solucions.

Un altre mètode multiagent també anomenat formigues fou introduit per Comellas i Ozón el 1995. En aquest cas les formigues es mouen directament sobre el graf que representa el problema. Aquí (i també en l'ACO), fent el paral·lelisme amb la colònia de formigues podem dir que cada una de les formigues actua de manera individual però els canvis s'obtenen de manera conjunta. És a dir, cada formiga actua millorant localment la solució però el resultat és una millora global sovint superior a si s'haguessin fet les individuals per separat.

1. Generar aleatoriament una solució inicial
2. Posar N formigues sobre la solució
3. Calcular els costos locals
4. Calcular el cost global
5. Repetir fins assolir el criteri d'aturada:
    a. Repetir per cada formiga:
        i. Moure amb probabilitat p_m al punt de la solució amb pitjor cost local, o amb probabilitat 1-p_m a qualsevol punt
        ii. Canviar amb probabilitat p_c la situació local per la millor situació possible, o amb probabilitat 1-p_c per una de qualsevol
        iii. Calcular els costos locals que hagin estat modificats
    b. Calcular el cost global
    c. Si és millor que la millor trobada fins el moment, guardar la nova solució

3.1.3.1. Funcionament

El funcionament de l'algorisme es basa en aplicar, com hem comentat anteriorment, una cerca paral·lela dels seus agents. Cada formiga que forma part del grup té unes variables individuals que les fa úniques dins de la colònia.

Les formigues venen determinades per un estat. Els diferents estats poden ser l'actual, on està la formiga; l'anterior, d'on prové la formiga i finalment el futur que és l'estat on la formiga es desplaçarà quan es dugui a terme un canvi d'estats. Per la realització d'aquests canvis fa falta, doncs, que les formigues es puguin moure sobre una estructura predefinida i estàtica que és coneguda des del començament i on s'hi poden ubicar.

Fer un canvi d'estats no és trivial per a la formiga ja que aquesta ha d'avaluar tots els possibles estats futurs per decidir quin d'ells és el que més convé segons uns certs paràmetres. En aquest punt queda definit el canvi que durà a terme la formiga. De totes maneres, tots els canvis depenen d'una probabilitat que fa que la formiga no faci el mateix tipus de moviments per evitar repeticions innecessàries i estancar l'algorisme.

L'èxit de l'algorisme però, està en un altre punt: com trobar les solucions parcials del problema. El que fan les formigues és realitzar canvis a l'estat actual en el que es troben de manera que, si aquests canvis milloren uns certs paràmetres que defineixen la relació entre paraules del codi, es donen per bons. D'altra manera es continua buscant una solució parcial satisfactòria.

En la següent figura podem veure representada l'acció d'una formiga:

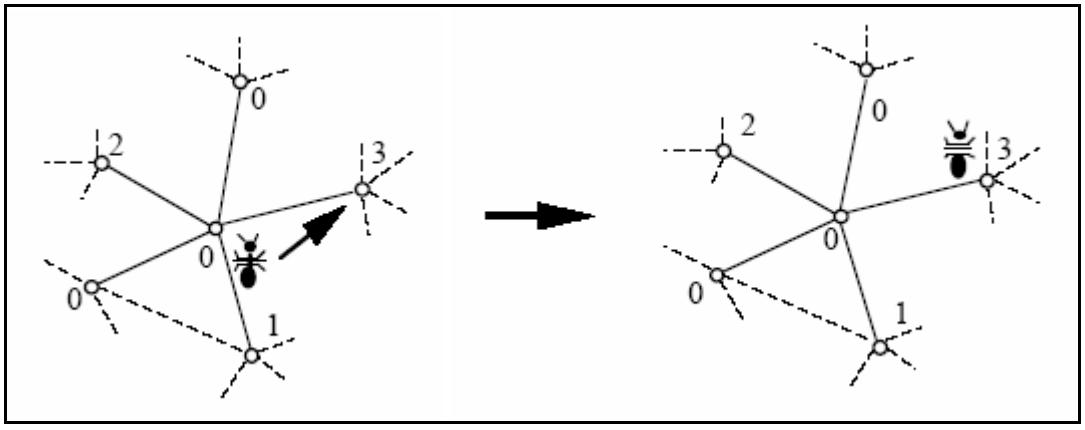

Fig.3.2 Acció que realitza una formiga quan canvia d'estat

Per tant, si ens mirem l'algorisme de les formigues com a conjunt, podem veure com uns agents es situen en diferents posicions i a cada pas i, depenent de la probabilitat associada, avancen cap a un lloc o cap a un altre, trobant en cada moment una solució parcial del problema, que no sempre serà la que volem.

Després de molts canvis d'estat s'ha de valorar si la solució final és realment la que esperàvem.

3.2. Generació de codis

Després de veure una petita introducció als diferents mètodes de recerca de codis, el que ara cal veure és què tenen en comú tots ells. Si bé és ben cert que tots compten amb matisos prou diferents, també ho és el fet que utilitzen els mateixos barems per determinar si les solucions són prou bones. Ens disposem doncs, a explicar la Funció de cost Local i la Funció de cost Global.

3.2.1. Funció de cost local

La funció de cost local és una funció que avalua el cost entre dues paraules del codi mitjançant la distància de Hamming corresponent. La podem definir de la següent manera:

$$\begin{aligned} &FCL(p_1, p_2) = \frac{1}{G^2 + 1} \\ &G = DH(p_1, p_2) \end{aligned} \qquad \text{on} \qquad \begin{cases} G \in \mathrm{N} \\ 0 < DH < \infty \\ p_1, p_2 \in C \end{cases} \qquad \textbf{(3.3)}$$

Com es pot veure per la definició, la FCL és inversament proporcional al quadrat de la distància de Hamming. Això vol dir que com més petita sigui aquesta, més gran serà la FCL i a la inversa.

La funció de cost local la utilitzarem bàsicament per el càlcul de la funció de cost global.

3.2.2. Funció de cost global

La funció de cost global està definida com:

$$FCG = \sum_{i=0}^{N-2} \left(\sum_{j=i+1}^{N-1} FCL(i, j) \right) \qquad \textbf{(3.4)}$$

on N és el número de paraules del codi

Aquesta funció la farem servir com a barem per discernir si la solució parcial que han trobat les formigues és acceptable o no. És a dir, si la FCG actual és menor que la FCG anterior vol dir que les formigues han fet uns canvis prou

bons que permeten que la funció de cost global es minimitzi i per tant en podem deduir que les DH seran, en general, més elevades.

3.2.3. Probabilitat

La probabilitat p_b és un factor que determina si la formiga ha d'avançar cap a un node futur amb DH dolenta comparada amb els altres possibles nodes futurs o per contra, la formiga ignorarà el node més desfavorit i anirà a qualsevol altre possible node futur. Com hem anat comentat al llarg d'alguns apartats, és de essencial que existeixi aquesta probabilitat. S'ha demostrat que els algorismes que sempre intenten millorar les pitjors circumstàncies, s'estanquen abans de trobar una solució. Per aquest motiu, l'algorisme de les formigues incorpora aquest camp que assegura que no es pugui entrar en cap mínim. Aquest concepte el podem visualitzar el les següents figures:

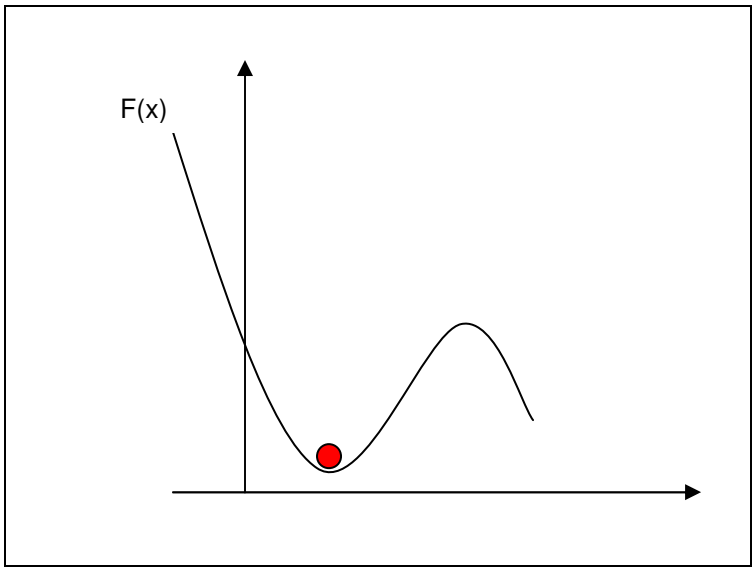

Fig.3.5 La formiga es troba en un mínim i no en surt perquè no existeix p_b

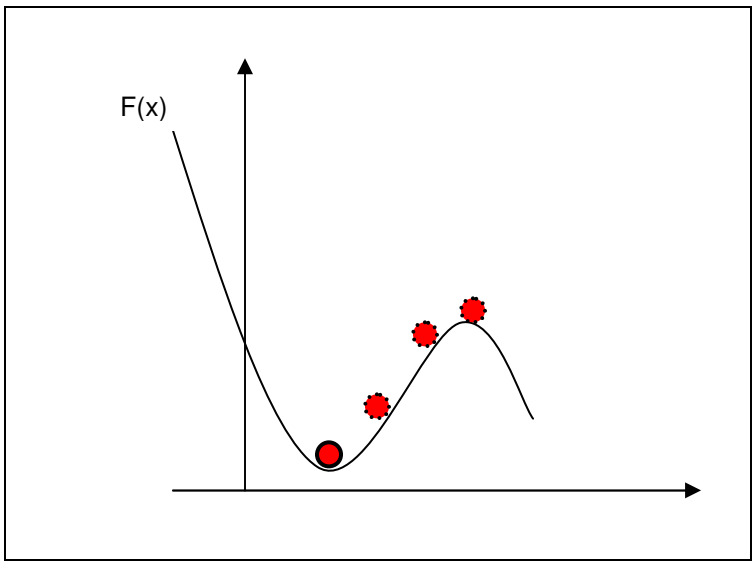

Fig.3.6 Formiga que es troba en un mínim i que gràcies a p_b és capaç de sortir

4. Algorisme formigues per a codis correctors d'errors

L'algorisme formigues actua sobre el graf construït a partir d'una de les dues topologies descrites abans i que està associat a un codi que ha estat generat per una combinació aleatòria de bits complint totes les condicions preestablertes.

4.1. Canvis que realitzen les formigues

Quan una formiga ha decidit mitjançant la DH el node futur es fa el canvi d'estats. El node futur passa a ser ara el node actual mentre que el node actual anterior passa a ser el node anterior. Evidentment el node futur encara està per definir. En aquest punt la formiga avalua si la DH entre aquests dos nodes és menor que la DH mínima que marca el codi. Si és així la formiga es disposa a realitzar uns certs canvis per tal de millorar aquesta DH i les relacionades amb aquestes paraules.

Els possibles canvis a realitzar són:

1.- Inversió dels bits
2.- Nova distribució

4.1.1. Inversió dels bits d'una paraula

La inversió dels bits d'una paraula consta de determinar quants dels bits uns que conté s'invertiran amb d'altres bits de la mateixa paraula que són zero. El màxim número de bits a invertir uns coincideix amb el pes de la paraula i la inversió sempre ha de fer com a mínim un canvi perquè sinó la paraula seria la mateixa.

Per tant, el que fem és agafar dues posicions aleatòries i mirar si en una hi ha un u i en l'altre hi ha un zero. Si és així es fa el canvi i sinó es continua buscant aquesta condició. Aquest procés és recursiu fins a un número qualsevol entre u i el pes de la paraula. Això és així per evitar que sempre es facin les mateixes inversions i el programa es pugui dinamitzar una mica.

Taula 4.1. Exemple d'inversió de bits

A (10,3,4) = 6		
Paraules	Pes aleatori	Paraules bits invertits
0100110010	2	*10*10010010
0101011000	4	0*0*10100*1*01
1101100000	1	*0*1011000*1*0

0100000111	3	1000101010
0100011010	2	0001001110
1100100001	4	0011000110

Les paraules invertides no han d'existir dins el codi ja que si fos així la distància de Hamming entre elles dues seria 0 i això no ha de poder passar.

4.1.2. Nova distribució dels bits d'una paraula

La nova distribució consta en tornar a gener aleatòriament una paraula que compleixi la condició que sigui única dins del codi (no pot estar repetida) així com les condicions de pes i de longitud.

Tant la inversió de bits com la nova distribució depenen d'un nombre aleatori n_a que indica quin dels dos canvis s'ha de produir. Encara que per defecte els canvis es realitzen amb la mateixa probabilitat, aquesta és un paràmetre més a tenir en compte a l'hora d'optimitzar l'algorisme.

4.2. Descripció dels passos d'una formiga

La formiga es col·loca de manera aleatòria en un node qualsevol per tal d'iniciar el seu recorregut. A partir d'aquí estableix a quin node futur anirà i posteriorment, quan hi sigui, decidirà a partir d'una probabilitat p_b quin canvi durà a terme. Per tant, podríem resumir l'activitat d'una formiga en els següents passos.

1.- Node inicial aleatori
2.- Decideix node futur
3.- Fa canvis:
  3.1.- Inversió de bits
  3.2.- Nova distribució
4.- Es repeteixen els passos 2 i 3 fins que s'arriba a una solució vàlida o concertada.

Es pot arribar a una solució concertada quan després d'haver fet un cert nombre de canvis (que sol ser molt elevat), la formiga no ha trobat cap resultat final apropiat. És a dir, no ha estat capaç de trobar un codi *A (n,d,w)* tal que es compleixi la distància de Hamming en totes les paraules. Si arriba un punt en què després de tantes iteracions el problema no queda resolt, les formigues avorten la missió.

5. RESULTATS

En aquest apartat mostrarem els resultats obtinguts utilitzant l'algorisme de les formigues tant en la modalitat de topologia aleatòria com en la topologia 1/3 per diferents distàncies de Hamming, així com les taules que ens mostren els resultats aconseguits fins al moment.

5.1. Distància de Hamming 4

5.1.1. Resultats Obtinguts

Taula 5.1. Paraules del codi A (10,4,7) = 13

	A (10,4,7) = 13
1	1111100011
2	0101111011
3	1111011010
4	0011111110
5	1011001111
6	1010110111
7	0110011111
8	1101110110
9	1101011101
10	0111110101
11	1011111001
12	1111101100
13	1100101111

Taula 5.2. Paraules del codi A (10,4,8) = 5

	A (10,4,8) = 5
1	1111101110
2	0111110111
3	1110011111
4	1001111111
5	1111111001

Taula 5.3. Paraules del codi A (11,4,8) = 17

	A (11,4,8) = 17
1	10111010111
2	01111110101
3	10111101110
4	11101100111

5	11110011101
6	01111001111
7	11011110011
8	01111111010
9	11111101001
10	11110110110
11	11101111100
12	00110111111
13	01001111111
14	10101111011
15	11010101111
16	11011011110
17	10011111101

Taula 5.4. Paraules del codi A (8,4,5) = 8

	A (8,4,5) = 8
1	00110111
2	10111100
3	01001111
4	11100110
5	10101011
6	11010101
7	11011010
8	01111001

Taula 5.5. Paraules del codi A (9,4,4) = 18

	A (9,4,4) = 18
1	010101001
2	111000010
3	100011001
4	000100111
5	100101010
6	011011000
7	011100100
8	000111100
9	101001100
10	001110010
11	110000101
12	001010101
13	010010011
14	010001110
15	001001011
16	110110000
17	100010110

18	101100001

Taula 5.6. Paraules del codi A (9,4,7) = 4

	A (9,4,7) = 4
1	110101111
2	101110111
3	111011011
4	111111100

5.1.2. Resultats Existents

	w=3	4	5	6	7	8	9
n=6	4^5						
7	7^5						
8	8^5	14^5					
9	12^9	18^9					
10	13^9	30^9	36^9				
11	17^9	35^9	66^9				
12	20^9	51^9	80- 84^9	132^9			
13	26^9	65^9	123- 132^9	166- 182^9			
14	28^9	91^9	169- 182^9	278- 308^9	325- 364^9		
15	35^9	105^9	237- 271^{13}	389- 455^9	585- 660^9		
16	37^9	140^9	315- 336^9	615- 722^9	836- 1040^9	1170- 1320^9	
17	44^9	156- 157^9	441- 476^9	854- 952^9	1416- 1753^9	1770- 2210^9	
18	48^9	198^9	518- 565^9	1260- 1428^9	2041- 2448^9	3186- 3944^9	3540- 4420^9

Fig.5.7 Resultats obtinguts fins ara on n és la longitud de la paraula, w el pes i on la DH és de 4 bits

5.2. Distància de Hamming 6

5.2.1. Resultats Obtinguts

Taula 5.8. Paraules del codi A (8,6,4) = 2

	A (8,6,4) = 2
1	01000111
2	10101100

Taula 5.9. Paraules del codi A (9,6,4) = 3

	A (9,6,4) = 3
1	010010110
2	100100101
3	001111000

Taula 5.10. Paraules del codi A (10,6,4) = 5

	A (10,6,4) = 5
1	0101110000
2	0110000101
3	1010100010
4	1000011100
5	0001001011

Taula 5.11. Paraules del codi A (10,6,5) = 6

	A (10,6,5) = 6
1	0110101010
2	1001011010
3	0001101101
4	0010010111
5	1111000100
6	1100110001

Taula 5.12. Paraules del codi A (11,6,4) = 6

	A (11,6,4) = 6
1	01100010010
2	00000011101
3	00110100100

4	10010000011
5	00001101010
6	11001000100

5.2.2. Resultats Existents

	w=4	5	6	7	8	9	10	11
n=8	2^{10}							
9	3^{5}							
10	5^{5}	6^{5}						
11	6^{5}	11^{5}						
12	9^{5}	12^{5}	22^{5}					
13	13^{5}	18^{21}	26^{9}					
14	14^{5}	28^{20}	42^{20}	42^{21}				
15	15^{5}	42^{9}	70^{20}	$69\text{-}78^{9}$				
16	20^{9}	48^{9}	112^{9}	$109\text{-}138^{9}$	$120\text{-}150^{20}$			
17	20^{21}	68^{9}	$112\text{-}136^{9}$	$166\text{-}228^{T}$	$184\text{-}280^{T}$			
18	22^{9}	$69\text{-}72^{9}$	$132\text{-}199^{T}$	$243\text{-}349^{9}$	$260\text{-}428^{20}$	$304\text{-}425^{20}$		
19	25^{21}	$76\text{-}83^{9}$	$172\text{-}228^{9}$	$338\text{-}520^{20}$	$408\text{-}718^{T}$	$504\text{-}789^{20}$		
20	30^{9}	$84\text{-}100^{9}$	$232\text{-}276^{9}$	$462\text{-}651^{9}$	$588\text{-}1107^{14}$	$832\text{-}1363^{20}$	$944\text{-}1403^{T}$	
21	31^{9}	$108\text{-}126^{9}$	$269\text{-}350^{9}$	$570\text{-}828^{9}$	$774\text{-}1695^{14}$	$1184\text{-}2359^{T}$	$1454\text{-}2685^{T}$	
22	37^{9}	$132\text{-}136^{9}$	$319\text{-}462^{9}$	$759\text{-}1100^{9}$	$1139\text{-}2277^{9}$	$1792\text{-}3766^{T}$	$2182\text{-}4415^{T}$	$2636\text{-}5064^{20}$
23	40^{9}	$147\text{-}170^{9}$	$399\text{-}521^{9}$	$969\text{-}1518^{9}$	$1436\text{-}3162^{9}$	$2271\text{-}5819^{9}$	$2970\text{-}7521^{20}$	$3585\text{-}7953^{20}$

Fig.5.13 Resultats obtinguts fins ara on n és la longitud de la paraula i w el pes i on la DH és de 6 bits

5.3. Distancia de Hamming 8

5.3.1. Resultats Obtinguts

Taula 5.14. Paraules del codi A (10,8,5) = 2

	A (10,8,5) = 2
1	0011110010
2	0100011101

Taula 5.15. Paraules del codi A (11,8,5) = 2

	A (11,8,5) = 2
1	00111001100
2	00000110111

Taula 5.16. Paraules del codi A (12,8,5) = 3

	A (12,8,5) = 3
1	010000001111
2	100110100100
3	101001011000

Taula 5.17. Paraules del codi A (12,8,6) = 4

	A (12,8,6) = 4
1	100101000111
2	011010100110
3	001111011000
4	110000111001

Taula 5.18. Paraules del codi A (13,8,5) = 3

	A (13,8,5) = 3
1	1110001000010
2	0000010110101
3	0001101001100

Taula 5.19. Paraules del codi A (13,8,6) = 4

	A (13,8,6) = 4
1	1010111001000

2	0100001101110
3	0011100110100
4	1101010010010

Taula 5.20. Paraules del codi A (14,8,5) = 4

	A (14,8,5) = 4
1	01011001000001
2	10000001101100
3	10010100010010
4	01100010000110

Taula 5.21. Paraules del codi A (15,8,5) = 6

	A (15,8,5) = 6
1	101100000010100
2	010000001010011
3	110000100101000
4	000101010001010
5	001010010100001
6	000011101000100

Taula 5.22. Paraules del codi A (16,8,5) = 6

	A (16,8,5) = 6
1	1100001000011000
2	0011001010000001
3	0110100001000100
4	0000000100101101
5	0000010111010000
6	0101010000100010

Taula 5.23. Paraules del codi A (17,6,5) = 6

	A (17,8,5) = 7
1	00000001001101001
2	11000010000000110
3	00100000010110010
4	10111000000001000
5	00010011110000000
6	00010100001010100
7	01001100100100000

5.3.2. Resultats Existents

	w=5	6	7	8	9	10	11	12
n=10	2^5							
11	2^{10}							
12	3^5	4^5						
13	3^{10}	4^{10}						
14	4^{10}	7^5	8^5					
15	6^5	10^5	15^5					
16	6^{10}	16^5	16^5	30^5				
17	7^{10}	17^5	24^{21}	34^9				
18	9^{10}	21^9	33- 39^9	46- 54^9	48- 68^9			
19	12^5	28^9	52- 57^9	78- 92^9	88- 114^9			
20	16^5	40^9	80^9	130- 142^9	160- 195^{20}	176- 228^9		
21	21^5	56^9	120^9	210^9	280- 320^{20}	336- 389^T		
22	21^{21}	77^9	176^9	330^9	280- 493^{14}	616- 641^{20}	672- 724^T	
23	23^5	77- 80^9	253^9	506^9	400- 742^T	616- 1078^T	1288- 1309^T	
24	24^5	78- 92^9	253- 274^9	759^9	640- 1078^T	960- 1624^T	1288- 2188^{20}	2576^{20}
25	30^9	100^9	254- 328^9	759- 856^9	829- 1539^T	1248- 2446^T	1662- 3554^T	2576- 4169^{20}

Fig.5.24 Resultats obtinguts fins ara on n és la longitud de la paraula i w el pes i on la DH és de 8 bits

5.4. Distància de Hamming 10

5.4.1. Resultats Obtinguts

Taula 5.25. Paraules del codi A (13,10,6) = 2

	A (13,10,6) = 2
1	1110110100000
2	0001101010110

Taula 5.26. Paraules del codi A (14,10,6) = 2

	A (14,10,6) = 2
1	01001001111000
2	10110010001010

Taula 5.27. Paraules del codi A (14,10,7) = 2

	A (14,10,7) = 2
1	01011001101010
2	10110110010010

Taula 5.28. Paraules del codi A (15,10,6) = 3

	A (15,10,6) = 3
1	010100100011010
2	001111010000001
3	100000011100110

Taula 5.29. Paraules del codi A (15,10,7) = 3

	A (15,10,7) = 3
1	100000101101101
2	010101110011000
3	011010001010110

Taula 5.30. Paraules del codi A (16,10,6) = 3

	A (16,10,6) = 3
1	0011100010011000
2	1100001010100001
3	0100010101000110

Taula 5.31. Paraules del codi A (17,10,6) = 3

	A (17,10,6) = 3
1	11100001100100000
2	00000110111001000
3	00011010000010110

Taula 5.32. Paraules del codi A (16,10,7) = 4

	A (16,10,7) = 4
1	0110110000110100
2	0011001100101010
3	1001011010000101
4	1000100111011000

Taula 5.33. Paraules del codi A (18,10,6) = 4

	A (18,10,6) = 4
1	000101100101000001
2	001010101000010100
3	000000010010100111
4	110001001010001000

Taula 5.34. Paraules del codi A (19,10,6) = 4

	A (19,10,6) = 4
1	0001011100000010100
2	1000010000011100001
3	0000000010100111010
4	0110101011000000000

Taula 5.35. Paraules del codi A (17,10,7) = 5

	A (17,10,7) = 5
1	10010011000111000
2	01000100001101011
3	01100011100000101
4	10011100110000010
5	00101000011110100

Taula 5.36. Paraules del codi A (20,10,6) = 5

	A (20,10,6) = 5
1	00101100000110010000
2	01100000000001100110
3	10000000110100000101
4	01010111010000000000
5	10000010001000011010

Taula 5.37. Paraules del codi A (21,10,6) = 7

	A (21,10,6) = 7
1	101000010000000110010
2	000000110011000001001
3	000100001000010000111
4	011110000010100000000
5	000011000000011011000
6	110001101100000000000
7	000000000101101100100

Taula 5.38. Paraules del codi A (22,10,6) = 7

	A (22,10,6) = 7
1	01000000001011011000000
2	10000011100100000000001
3	01100000100000000011010
4	00001010010000001000110
5	00010100001100000010100
6	10101100000000010100000
7	00010001010001100010000

5.4.2. Resultats Existents

	w=6	7	8	9	10	11	12	13
n=12	2^5							
13	2^5							
14	2^{10}	2^{10}						
15	3^5	3^5						
16	3^{10}	4^5	4^{10}					
17	3^{10}	5^5	6^5					
18	4^{10}	6^5	9^5	10^5				
19	4^{10}	8^5	12^{10}	19^5				
20	5^{10}	10^{10}	17^{21}	20^5	38^5			
21	7^5	13^{11}	21^5	27- 35^9	38- 42^9			
22	7^5	16^{21}	24- 33^9	35- 51^9	46- 73^{21}	46- 81^{21}		
23	8^5	20^{21}	33- 46^9	45- 81^{20}	54- 117^9	65- 135^{20}		
24	9^{10}	24^5	38- 60^9	56- 119^{20}	72- 171^{20}	95- 223^{20}	122- 247^{20}	
25	10^{10}	28- 32^9	48- 75^9	72- 158^{20}	100- 262^{20}	125- 383^T	132- 444^T	
26	13^5	28- 36^{14}	54- 104^9	91- 214^{20}	130- 410^9	168- 581^{20}	195- 728^{20}	210- 824^T
27	14^{10}	36- 48^{14}	66- 121^9	118- 299^{20}	162- 577^9	222- 900^{20}	351- 1289^{20}	405- 1460^{20}

Fig.5.39 Resultats obtinguts fins ara on n és la longitud de la paraula i w el pes i on la DH és de 10 bits

5.5. Distància de Hamming 12

5.5.1. Resultats Obtinguts

Taula 5.40. Paraules del codi A (14,12,7) = 2

	A (14,12,7) = 2
1	00011101110010
2	10100011001101

Taula 5.41. Paraules del codi A (15,12,7) = 2

	A (15,12,7) = 2
1	010011001110001
2	001100100001111

Taula 5.42. Paraules del codi A (16,12,7) = 2

	A (16,12,7) = 2
1	0111001010101000
2	0001110100010110

Taula 5.43. Paraules del codi A (16,12,8) = 2

	A (16,12,8) = 2
1	0111011100011000
2	1000100011110110

Taula 5.44. Paraules del codi A (17,12,7) = 2

	A (17,12,7) = 2
1	11001001000100110
2	00110010110000011

Taula 5.45. Paraules del codi A (17,12,8) = 2

	A (17,12,8) = 2
1	11010110010001001
2	00101011011100100

Taula 5.46. Paraules del codi A (18,12,7) = 3

	A (18,12,7) = 3
1	010001110000100110
2	010000001110011001
3	101110010011000000

Taula 5.47. Paraules del codi A (18,12,8) = 3

	A (18,12,8) = 3
1	000001111001110010
2	110010110110001000
3	001111000010100101

Taula 5.48. Paraules del codi A (18,12,9) = 4

	A (18,12,9) = 4
1	101011010010101010
2	000110011101011100
3	111100101100100001
4	010001100011010111

Taula 5.49. Paraules del codi A (19,12,7) = 3

	A (19,12,7) = 3
1	0001010110001001100
2	0111000001100110000
3	1100101100010000010

Taula 5.50. Paraules del codi A (19,12,8) = 3

	A (19,12,8) = 3
1	0110011001101001000
2	0001110100000110110
3	1000100110110001001

Taula 5.51. Paraules del codi A (19,12,9) = 4

	A (19,12,9) = 4
1	1010010001100111001
2	1001100010111010100
3	0111011011001000010
4	0100001100010101111

Taula 5.52. Paraules del codi A (20,12,7) = 3

	A (20,12,7) = 3
1	00010001011001010100
2	01110000000000101011
3	10000100100110011000

Taula 5.53. Paraules del codi A (20,12,8) = 5

	A (20,12,8) = 5
1	11100010000101011000
2	01010011001000100110
3	00100101110010001010
4	10011100100000010101
5	00001000011111100001

Taula 5.54. Paraules del codi A (20,12,9) = 5

	A (20,12,9) = 5
1	11010000111010001001
2	00110011100101100001
3	00100110011100001110
4	10001101100110010010
5	01001001010001111100

Taula 5.55. Paraules del codi A (21,12,7) = 3

	A (21,12,7) = 3
1	10010111000100000 1000
2	01100000101011100000
3	00101000010100001 0011

Taula 5.56. Paraules del codi A (22,12,7) = 4

	A (22,12,7) = 4
1	0000000100011010101010
2	1000010011010001000100
3	0011110100100000010000
4	0100001001100110000001

Taula 5.57. Paraules del codi A (23,12,7) = 4

	A (23,12,7) = 4
1	00001000110100000010011
2	01000000101000100101100
3	10010001000010010001001
4	00100110100011001000000

Taula 5.58. Paraules del codi A (24,12,7) = 4

	A (24,12,7) = 4
1	001010000101001110000000
2	000000110001110001001000
3	010100000110000000010011
4	000011001000100000100110

Taula 5.59. Paraules del codi A (25,12,7) = 5

	A (25,12,7) = 5
1	0000000001100001001001110
2	1011100010000010000000010
3	0001010101010000000010001
4	0000011010000100011100000
5	0100000100001010110000100

Taula 5.60. Paraules del codi A (26,12,7) = 5

	A (26,12,7) = 5
1	11001001010000000011000000
2	00000100001010100001001001
3	00000010100100011010001000
4	01010000000010010100000110
5	10100000000101100000110000

Taula 5.61. Paraules del codi A (27,12,7) = 6

	A (27,12,7) = 6
1	000000001001001101010000100
2	101001000100000000010101000
3	010011000010100000100000100
4	001000111000000010100000010
5	110100100000010001000010000
6	000010010000011000001100001

5.5.2. Resultats Existents

	w=7	8	9	10	11	12	13	14
n=14	2^5							
15	2^5							
16	2^5	2^{10}						
17	2^{10}	2^{10}						
18	3^5	3^5	4^5					
19	3^5	3^{10}	4^5					
20	3^{10}	5^5	5^5	6^5				
21	3^{10}	5^5	7^5	7^5				
22	4^5	6^5	8^5	11^5	12^5			
23	4^{10}	6^{10}	10^{10}	16^{10}	23^5			
24	4^{10}	9^5	16^5	24^5	24^5	46^5		
25	5^{10}	10^5	25^5	28- 38^{20}	36- 42^9	50^9		
26	5^{10}	13^5	26^5	33- 37^T	39- 69^{21}	54- 83^{21}	58- 92^{21}	
27	6^{10}	15^{10}	39^9	39- 58^B	54- 90^B	82- 140^{20}	86- 156^{20}	
28	8^5	19^{11}	39- 45^{20}	49- 87^T	65- 147^B	84- 199^{20}	99- 245^{20}	172- 265^{20}

Fig.5.62 Resultats obtinguts fins ara on n és la longitud de la paraula i w el pes i on la DH és de 12 bits

5.6. Discussió dels resultats

Com hem pogut anar veient, els resultats obtinguts s'igualen en la majoria de casos als resultats existents. Tot i no disposar d'una màquina potent podem dir que resultats més complexes s'obtenen després de moltes hores acumulades de càlcul computacional. Per tant de moment ens hem basat més en veure si aplicant l'algorisme de les formigues es podia igualar els resultats bàsics existents fins el moment.

Si comparem les taules que hem vist anteriorment amb els resultats obtinguts podem veure que l'algorisme presenta unes bones prestacions, ja que a més d'aconseguir el resultat esperat el cost temporal també és molt petit. Podem dir, doncs, que per casos relativament simples l'algorisme té una bona eficiència. Per altra banda, quan es tracta de casos complicats, on s'ha demostrat el límit teòric de paraules, l'algorisme no es capaç de trobar una solució amb un cost temporal relativament baix. Segurament es podria obtenir la solució però a canvi d'incrementar notòriament el temps de càlcul computacional.

També cal dir que les dues topologies donen eficiències diferents. Pel que fa a la topologia aleatòria és capaç de crear una xarxa que al ser aleatòria proporciona propietats molt bones. Per un costat és molt difícil trobar la solució el primer cop que s'executa l'algorisme, però per l'altra, al ser una topologia dinàmica, s'incrementa altament la probabilitat que trobi una solució satisfactòria. En canvi, pel que fa a la topologia 1/3, al ser estàtica provoca un estancament de les formigues que no són capaces de trobar (en molts casos) la solució.

Pel que fa als canvis que realitza una formiga quan està en un node, cal remarcar que hem notat una millora en el cost temporal si donen més probabilitat d'ús a la inversió de bits envers la nova distribució. El cas és que una nova distribució que crea una nova paraula té més probabilitat de que la paraula ja existeixi i això repercuteix directament amb un empitjorament del temps emprat per trobar una solució.

La probabilitat que té una formiga d'anar al node futur que anteriorment ha decidit també influeix en certa manera al resultat final que ens dóna. Hem comprovat experimentalment que la formiga actua més eficaçment si la probabilitat és troba entre $0.55<p_b<0.80$. No es tracta de donar un valor exacte. De fet, cada càlcul requereix un ajust d'aquesta probabilitat ja que canvia el nombre de paraules, el nombre de bits de cada paraula, el pes i la distància de Hamming.

I finalment, la última variable que tenim en aquest algorisme és el número de formigues. Les proves que hem anat realitzant durant tot aquest temps ens han demostrat que un a partir d'un número de formigues concret l'algorisme deixa de millorar la seva eficàcia. Pot passar que el número de formigues per un cas concret sigui insuficient i això ens tradueix en un augment del cost temporal, però encara és més probable que un número massa elevat de formigues faci estancar més ràpidament l'algorisme. Per això és important utilitzar un terme mig. En ser coherent n'hi ha prou.

6. IMPLEMENTACIÓ

La implementació de l'algorisme de les formigues sobre codis correctors d'errors s'ha dut a terme amb el llenguatge de programació C. S'ha triat aquest i no un altre per dur-la a terme perquè ens interessava que el programa fos versàtil per tal que amb pocs canvis es pogués executar sobre qualsevol plataforma. D'aquesta manera el que s'ha intentat ha estat adaptar-lo per tal de poder-lo executar en una màquina d'alt rendiment linux del departament de Matemàtica IV de la Universitat.

Tant el codi com l'executable es troben dins del CD que s'adjunta en aquest projecte.

7. CONCLUSIONS

Un dels objectius del projecte era comprovar la possible validesa del mètode formigues per a la generació de codis correctors d'errors. Després de l'adaptació de l'algorisme a la cerca de codis, la implantació pràctica en C i la realització de proves en profunditat, de l'anàlisi dels resultats, podem dir que el mètode és tan vàlid com per exemple, Simulated annealing o els algorismes genètics. Malgrat l'existència de diferències importants entre ells, hem comprovat que tots són capaços de trobar solucions al problema plantejat tot i utilitzar metodologies diferents.

També cal dir que per a què els resultats d'aquest projecte es facin una mica més rellevants caldria executar *ALFO* en màquines computacionalment més potents que siguin capaces de trobar en un temps determinat una solució.

Pel que fa a l'impacte medi ambiental degut a la realització del projecte és el corresponent a l'ús d'un ordinador. D'altra banda també cal remarcar que l'impacte medi ambiental corresponent a la possible aplicació dels resultats obtinguts és positiu, ja que l'utilització dels codis correctors permet que es pugui transmetre la informació de forma eficient i així un estalvi d'energia.

8. BIBLIOGRAFIA

Format Electrònic

- [1] Agrell E., Vardy A. i Zeger K., "Tables of Binary Blocks Codes", [25 /04/2005], <http://www.s2.chalmers.se/~agrell/bounds/>

- [2] Comellas F., Abril J., Cortés A., Ozón J. i Vaquer M., "A multiagent System for Frequency Assignment in Cellular Radio Networks", *IEEE Trans. Vehic. Technology* , 49 (5), 1558-1565. (2000). [21/03/ 2005], <http://www-mat.upc.es/~comellas/freqassig/ieee-vt.pdf>

- [3] Knight D.G., "Some lower bounds for constant-weight codes", [23/04/2005], <http://www.glam.ac.uk/sot/Staff/andw.doc>

- [4] MacKay D., "David MacKay's Gallager code resources", [28/02/2005], <http://www.inference.phy.cam.ac.uk/mackay/CodesFiles.html>

- [5] Rains E. M. i Sloane N. J. A., "Table of Constant Weigh Binary Codes ", [22/04/2005], <http://www.research.att.com/~njas/codes/Andw/ >

Format paper

- [6] Agrell E., Vardy A., Zeger K., "Upper bounds for constant-weight codes.", *IEEE Transactions on Information Theory*, 46 (7) , 2373 – 2395 (2000)

- [7] Alspector J, Goodman R. and Brown T.X., "Applications of Neural Networks to Telecommunications", (Eds.) Lawrence Erlbaum Ass., Inc., Publis., Hillsdale, NJ (1993), pp.119-124; ISBN 0-8058-1560-0.

- [8] Brouwer A.E., Shearer J.B., Sloane N.J.A., Smith W.D., "A new table of constant weight codes", *IEEE Transactions on Information Theory*, 36 (6), 1334 – 1380 (1990).

- [9] Comellas, F. i Roca R. "Using genetic algorithms to design constant weight codes." *Applications of Neural Networks to Telecommunications* J. Alspector, R. Goodman and T.X. Brown (Eds.), Lawrence Erlbaum Ass., Inc., Publis., Hillsdale, NJ (1993), pp.119-124.

- [10] Nurmela K.J., Kaikkonen M.K., Ostergard, P.R.J, "New constant weight codes from linear permutation groups"; *IEEE Transactions on Information Theory*, 43 (5), 1623 – 1630 (1997)

9. ANNEX

L'annex que es troba dins el CD-Rom que s'adjunta amb aquest Treball Fi de Carrera conté el programa *ALFO* en llenguatge C. També inclou una carpeta amb els resultats obtinguts mitjançant el programa (carpeta de resultats).